

**FIG. 1**

```
1  /*The _CompFrameworkInterface contains interface for communications between
2   * the component framework and the components. The component framework uses this
3   * data structure to manage and communicate with components. The components use this
4   * data structure to publish and/or remove communication interfaces. Also, the components use
5   * this data structure to register listeners that listen to supported events (see below).
6   * The components initialize the following members when this structure is declared:
7   * 1) getName
8   * 2) getVersion
9   * 2) init
10  * 4) replace
11  * 5) run
12  * 6) stop
13  *
14  * The component framework initializes the following members when it retrieves
15  * a pointer to an instance of this structure from the components:
16  * 1) publish
17  * 2) remove
18  * 2) retrieve
19  * 4) addListener
20  * 5) removeListener
21  */
22  typedef struct _CompFrameworkInterface {
```

**FIG. 2A**

```

1      /*-----*
2      ** IDENTIFICATION AND VERSIONING OF COMPONENTS **
3      **-----*
4      ** Components must initialize these two members when *
5      ** an instance of this structure is declared.          *
6      **-----*/
7      /*
8      * const char* getName(void) - Returns the name of the component.
9      * The component framework uses this method to identify and manage the component.
10     */
11     const char* (*getName)(void);
12     /*
13     * const char* getVersion(void) - Returns the version of this component. The component
14     * framework uses this method to identify and to manage the component.
15     */
16     const char* (*getVersion)(void);
17
18

```

**FIG. 2B**

```

1  /*-----*
2  ** LIFETIME MANAGEMENT OF COMPONENTS      *
3  **-----*
4  ** Components must initialize these two members when *
5  ** an instance of this structure is declared.      *
6  **-----*/
7  /*
8  * init(void *initData) - This function shall be invoked by the
9  * component framework to give the component a chance to initialize.
10 *
11 * INPUT:
12     * initData - Points to the data a component uses to initialize itself. If this argument
13     * is NULL, no data available. The initData argument is generally available when one
14     * component replaces another component. The initData comes from the 'to be replaced
15     * component' via the replace() function.
16     * Return:
17     * 0 - Success
18     * -n - Error code
19 */
20 int (*init)(void* initData);
21 /*
22 replace(void) - This function shall be invoked by the component framework to notify a

```

**FIG. 2C**

1 \* running component that it is being replaced by another component. The running component  
2 \* must either return a NULL pointer or a pointer to the data that the new component uses  
3 \* to initialize itself. The type of the returned data is upto the components to comprehend.  
4 \* The component framework does not dictate any types.  
5 \*  
6 \* Here is how the replacement of component process works:  
7 \* 1) The component framework receives a "Replace" command.  
8 \* 2) The component framework invokes the replace() function of the to-be-replaced  
9 \* component.  
10 \* 2) The to-be-replaced component returns a pointer (can be NULL) to the data that is  
11 \* used by the new component to initialize itself.  
12 \* 4) The component framework invokes the init() functions of the new component passing  
13 \* it the returned pointer.  
14 \* 5) The component framework invokes the stop() function of the to-be-replaced  
15 \* component.  
16 \* 6) 100 milliseconds after the invocation of the stop() function, the component framework  
17 \* invokes the run() function of the new component  
18 \* 7) The component framework generates a COMPONENT\_STOPPED event.  
19 \* 8) The component framework generates a COMPONENT\_STARTED event.  
20 \*  
21 \* INPUT:  
22 \* initData - Points to the data a component uses to initialize itself. If this argument is

**FIG. 2D**

```

1      * NULL, no data available. The initData argument is generally available when one
2      * component replaces another component. The initData comes from the 'to be replaced
3      * component' via the replace() function.
4      * Return:
5      * 0 - Success
6      * -n - Error code
7      */
8      void* (*replace)(void);
9      /*
10     * run(void *anyData) - This function shall be invoked by the component framework to
11     * indicate that it is now safe for the component to perform normal processing.
12     *
13     * This function is conceptually equivalent to the main() function in procedural programming.
14     * This run() function is called ONCE by the component framework.
15     *
16     * INPUT:
17     * argc - The number of command line arguments.
18     * argv - The command line arguments. It is safe for components to keep a pointer to this
19     * list of arguments. Note: DO NOT deallocate/free the memory used by this argument.
20     *
21     * Return:
22     * 0 - Success

```

**FIG. 2E**

```
1      * -n - Error code
2      */
3      int (*run)(int argc, char** argv);
4      /*
5          stop(void) - This function shall be invoked by the component framework when it receives a
6          * Stop Component, Stop All or Shutdown command. This method is also invoked when the
7          * framework is about to shutdown regardless of reasons.
8          *
9          * Return:
10         * 0 - Success
11         * -n - Error code
12         */
13      int (*stop)(void);
14
15
16
```

**FIG. 2F**

```

1  /*-----*
2  ** COMMUNICATION INTERFACE PUBLICATION AND RETRIEVAL  *
3  **-----*
4  ** The publish() and retrieve() members (pointer to functions) are used by components for
5  ** inter-component communications. The producer components use the publish() function
6  ** to publish or circulate one or more communication interfaces for other components to use.
7  ** The consumer components retrieve the published interfaces via the retrieve() function.
8  ** The remove() function is the reverse of the publish() function. That is, to remove or
9  ** a published interface.
10 **
11 ** The component framework initializes the three member functions below
12 ** immediately after a pointer of this data structure (CompFrameworkInterface)
13 ** is retrieved from a component. component framework initializes:
14 ** 1) publish 2) retrieve 2) remove
15 **
16 ** Components must initialize the above three members to NULL when initialize this data
17 ** structure while declaring it. Otherwise, just leave them alone until they are ready for uses
18 ** (see below).
19 **
20 ** Components can invoke these three functions after or during the init() function
21 ** (the init member) is invoked. The system will CRASH if these publish(), retrieve(),
22 ** and remove() functions are invoked before the invocation of the init() function.

```

**FIG. 2G**



```

1  ** The component framework invokes the init() function.
2  **-----*/
3  /*
4  * publish() is used by the producer component to publish/publish an interface for consumer
5  * components to retrieve and communicate with it. The producer component can remove
6  * that published interface at anytime after its publish.
7  *
8  * NOTE: Do NOT invoke this function before the init() function is invoked by the component
9  * framework. This publish() function can be invoked in the init() function.
10 *
11 * interfaceName plus interfaceVersion must be UNIQUE throughout the system.
12 *
13 * interfaceName - The name of the interface. It can be different from the name of the
14 * component.
15 * interfaceVersion - The name of the interface. It can be different from the version of the
16 * component.
17 * commInterface - Points to the interface the producer component wants consumer
18 * components to use to communicate with it. This interface is retrieved by
19 * the retrieve() function.
20 *
21 * Return:
22 * 0 - Success

```

**FIG. 2H**

```

1      * -n - Error code
2      */
3      int (*publish)(const char* interfaceName, const char* interfaceVersion,
4                     void* commInterface);
5      /*
6      * remove() is used by the producer component to remove a published interface from further
7      * uses by consumer components.
8      *
9      * NOTE: Do NOT invoke this function before the init() function is invoked by the component
10     * framework. This remove() function can be invoked in the init() function.
11     *
12     * interfaceName - The name of the interface. It can be different from the name of the
13     * component.
14     * interfaceVersion - The name of the interface. It can be different from the version of the
15     * component.
16     * Return:
17     * 0 - Success
18     * -n - Error code
19     */
20     int (*remove)(const char* interfaceName, const char* interfaceVersion);
21     /*
22     * retrieve() is used by the consumer components to retrieve a published interface. Each

```

**FIG. 2I**

```

1      * invocation of this function returns the specified published interface if it exists.
2      *
3      * NOTE: Do NOT invoke this function before the init() function is invoked by the component
4      *       framework. This retrieve() function can be invoked in the init() function.
5      *
6      * interfaceName - The name of the published interface. It can be different from the name of
7      *                   the component.
8      * interfaceVersion - The name of the published interface. It can be different from the version
9      *                     of the component.
10     *
11     * Return:
12     *   A pointer to the interface the producer component wants consumer components to use to
13     *   communicate with it. Otherwise, a NULL pointer is returned if the specified interface is
14     *   not found.
15     */
16     void* (*retrieve)(const char* interfaceName, const char* interfaceVersion);
17

```

**FIG. 2J**

```

1  /*-----*
2  ** NOTIFICATION OF EVENTS *
3  **-----*
4  ** The following functions are used to inform the registered entities when components
5  ** started or stopped, or when an interface is published or removed. An event is generated
6  ** generated by the component framework when a component completely started or
7  ** stopped. Any components that interest in those events can register with the
8  ** component framework to be notified when those events occur.
9  **
10 ** The component framework initializes those two members immediately after a pointer
11 ** pointer of this data structure (CompFrameworkInterface) is retrieved from a component.
12 **-----*/
13 /*
14  * addListener() registers or adds the specified listener that listens to the specified
15  *      event (evt_type).
16  * evt_type - The event the specified listener listens.
17  * componentName - The name of the component this listener belongs.
18  * componentVersion - The version of the component this listener belongs.
19  * listener - The function to be invoked when the specified event occurs.
20  * eventData - Points to the data structure containing information about the occurred event.
21  *
22  * Return:

```

**FIG. 2K**

```

1      * 0 - Success
2      * -n - Error code
3      */
4      int (* addListener)(EventType evt_type,
5                          const char* componentName,
6                          const char* componentVersion,
7                          void (*listener)(EventDesc *eventData));
8      /*
9      * removeListener() removes the specified listener from listening to the specified event.
10     * evt_type - The event the specified listener listens.
11     * componentName - The name of the component this listener belongs.
12     * componentVersion - The version of the component this listener belongs.
13     * listener - The function to be removed or unregistered.
14     * eventData - Points to the data structure containing information about the occurred event.
15     *
16     * Return:
17     * 0 - Success
18     * -n - Error code
19     */
20     int (* removeListener)(EventType evt_type,
21                             const char* componentName,
22                             const char* componentVersion,

```

**FIG. 2L**

```
1         void (*listener)(EventDesc *eventData));  
2     } CompFrameworkInterface;  
3  
4     #endif
```

**FIG. 2M**

```

1  /* The following enumeration is used to indicate the type of event. */
2  typedef enum {
3      COMPONENT_STARTED,  /* a component was started. */
4      COMPONENT_STOPPED,  /* a component was stopped. */
5      INTERFACE_ISSUED,   /* an interface was published/published. */
6      INTERFACE_REMOVED,  /* an interface was removed. */
7      COMMAND_ISSUED,     /* an administrative command was registered with the system. */
8  } EventType;
9
10 /*
11 The _EventDesc structure contains information describing the following events that are related
12 * to components and interfaces:
13 * 1) ComponentStarted - When a component is started.
14 * 2) ComponentStopped - When a component is stopped.
15 * 3) InterfaceIssued - When an interface is published/published.
16 * 4) InterfaceRemoved - When an interface is removed.
17 */
18
19 typedef struct _EventDesc {
20     /* The name of the component/interface associated with this event.
21     * If the eventType is either COMPONENT_STARTED or COMPONENT_STOPPED,
22     * then name refers to the name of the concerned component. If

```

**FIG. 3A**

```

1      * the eventType is either INTERFACE_ISSUED or INTERFACE_REMOVED,
2      * then the name refers to the name of the concerned interface.
3
4      * If the event type is COMMAND_ISSUED, this member variable contains the
5      * entered/published command. */
6      char *name;
7
8      /* The version of the component/interface associated with this event.
9      * If the eventType is either COMPONENT_STARTED or COMPONENT_STOPPED,
10     * then version refers to the version of the concerned component.
11     *
12     * If the eventType is either INTERFACE_ISSUED or INTERFACE_REMOVED,
13     * then the version refers to the version of the concerned interface.
14     *
15     * If the event type is COMMAND_ISSUED, this member variable contains a NULL pointer.
16     * That is, (char *)NULL. */
17     char *version;
18
19     /* The type of this event. */
20     EventType type;
21
22     /* When did this event occur? */

```

**FIG. 3B**



```
1
2     time_t whenOccurred;
3
4 } EventDesc;
5
1
```

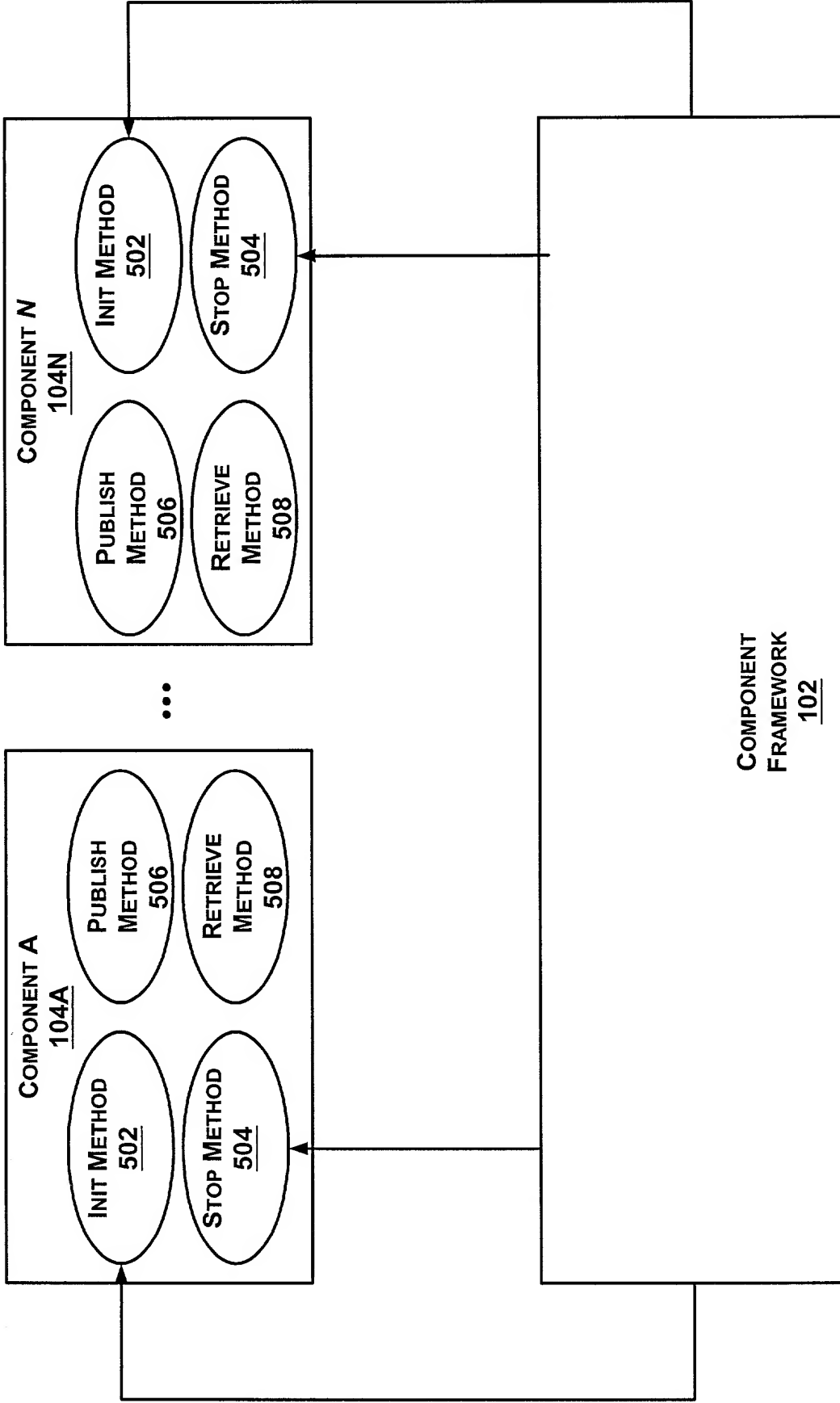
**FIG. 3C**

```

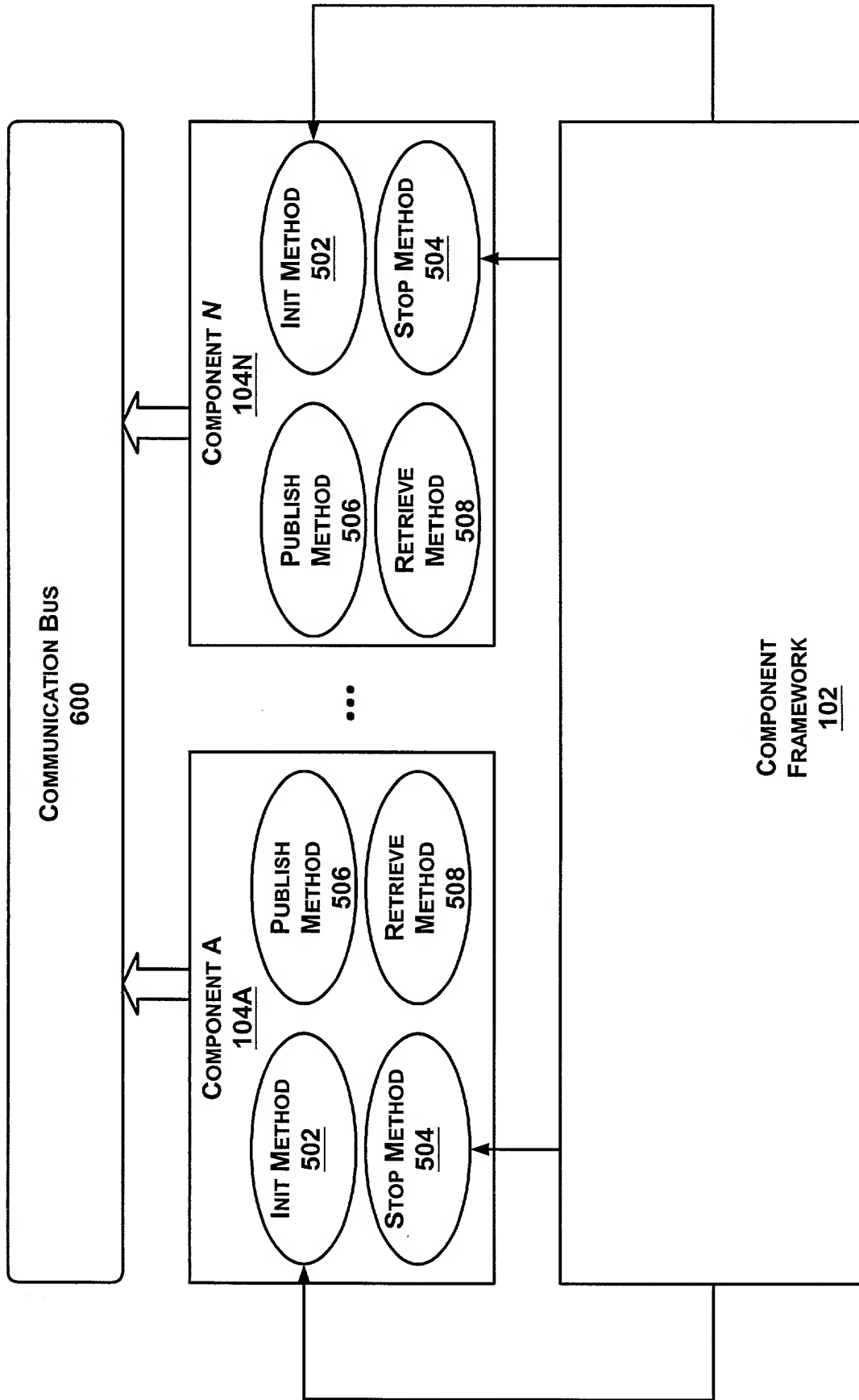
1  /* Definitions related to message queue. These definitions are used by external entities which
2  * wish to communicate with the component framework and the running components. */
3
4  #define CONFIG_MSG_Q_KEY  12764
5  #define INCOMING_MSG_TYPE  100
6  #define OUTGOING_MSG_TYPE  200
7  #define MSG_BODY_SIZE     256
8
9  /*
10 Data structure related to message queue. This structure is used by external entities which
11 * wish to communicate with the component framework and the running components.
12 */
13
14 typedef struct _MsgBuffer {
15     long msgType;          /* Incoming msg. type. */
16     long respondMsgType;   /* Outgoing msg. type. */
17     char msgBody[MSG_BODY_SIZE]; /* E.g. StartComponent mycompo 1.2.3.4 */
18 } MsgBuffer;
19
19
19
19
19
19
19
19

```

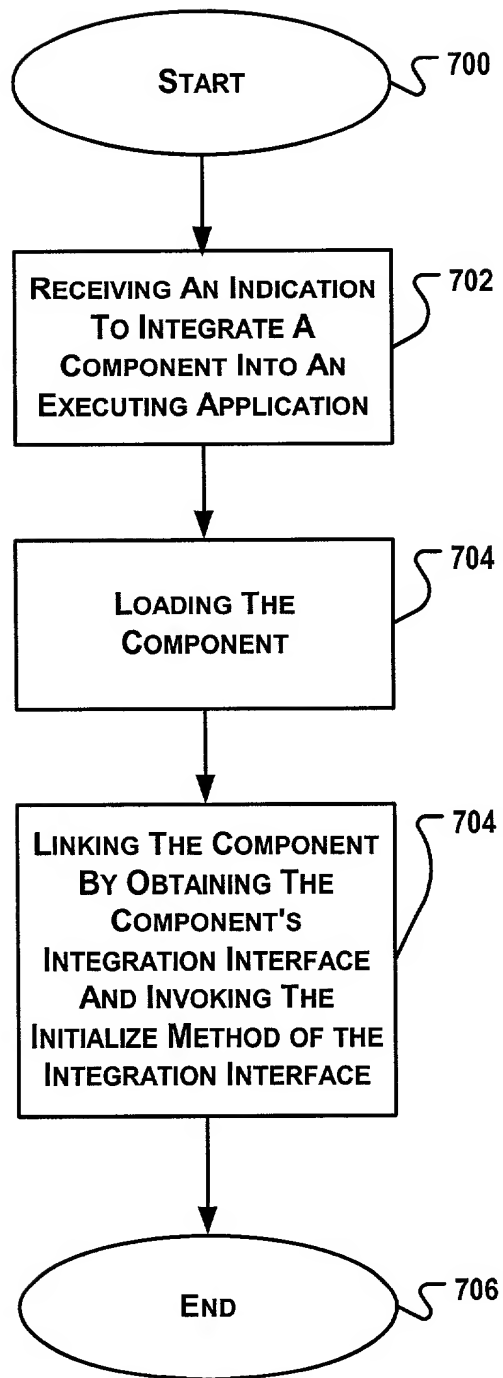
**FIG. 4**



**FIG. 5**



**FIG. 6**



**FIG. 7**